Parameterized Streaming Algorithms for Min-Ones d-SAT

Akanksha Agrawal
Department of Computer Science,
Ben-Gurion University of the Negev,
Beer-Sheva, Israel
akanksha.agrawal.2029@gmail.com

Arindam Biswas
The Institute of Mathematical Sciences, HBNI,
Chennai, India
arindam.b@ftml.net

Édouard Bonnet
CNRS, ENS de Lyon,
Université Claude Bernard Lyon 1
LIP UMR5668, France
edouard.bonnet@ens-lyon.fr

Nick Brettell
Department of Computer Science,
Durham University, Durham, UK
nbrettell@gmail.com

Radu Curticapean
BARC, University of Copenhagen,
Copenhagen, Denmark
ITU Copenhagen, Copenhagen, Denmark
radu.curticapean@gmail.com

Dániel Marx
Institute for Computer Science and Control,
MTA SZTAKI, Budapest, Hungary
dmarx@cs.bme.hu

Tillmann Miltzow
Department of Computer Science,
Utrecht University, Utrecht, Netherlands
t.miltzow@googlemail.com

Venkatesh Raman
The Institute of Mathematical Sciences, HBNI,
Chennai, India
vraman@imsc.res.in

Saket Saurabh
The Institute of Mathematical Sciences, HBNI,
Chennai, India
Department of Computer Science,
University of Bergen, Bergen, Norway
saket@imsc.res.in

Abstract

In this work, we initiate the study of the Min-Ones d-SAT problem in the parameterized streaming model. An instance of the problem consists of a d-CNF formula F and an integer k, and the objective is to determine if F has a satisfying assignment which sets at most k variables to 1. In the parameterized streaming model, input is provided as a stream, just as in the usual streaming model. A key difference is that the bound on the read-write memory available to the algorithm is $\mathrm{O}(f(k)\log n)$ ($f : \mathbb{N} \to \mathbb{N}$, a computable function) as opposed to the $\mathrm{O}(\log n)$ bound of the usual streaming model. The other important difference is that the number of passes the algorithm makes over its input must be a (preferably small) function of k.

We design a $(k+1)$-pass parameterized streaming algorithm that solves Min-Ones d-SAT ($d \geq 2$) using space $\mathrm{O}\big((kd^{ck} + k^d)\log n\big)$ ($c > 0$, a constant) and a $(d+1)^k$-pass algorithm that uses space $\mathrm{O}(k \log n)$. We also design a streaming kernelization for Min-Ones 2-SAT that makes $(k+2)$ passes and uses space $\mathrm{O}\big(k^6 \log n\big)$ to produce a kernel with $\mathrm{O}\big(k^6\big)$ clauses.

To complement these positive results, we show that any k-pass algorithm for Min-Ones d-SAT ($d \geq 2$) requires space $\Omega\big(\max\big\{n^{1/k}/2^k, \log(n/k)\big\}\big)$ on instances (F, k). This is achieved via a reduction from the streaming problem POT Pointer Chasing (Guha and McGregor [ICALP 2008]), which might be of independent interest. Given this, our $(k+1)$-pass parameterized streaming algorithm is the best possible, inasmuch as the number of passes is concerned.

In contrast to the results of Fafianie and Kratsch [MFCS 2014] and Chitnis et al. [SODA 2015], who independently showed that there are 1-pass parameterized streaming algorithms for Vertex Cover (a restriction of Min-Ones 2-SAT), we show using lower bounds from Communication

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 8; pp. 8:1–8:20

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Complexity that for any $d \geq 1$, a 1-pass streaming algorithm for Min-Ones d-SAT requires space $\Omega(n)$. This excludes the possibility of a 1-pass parameterized streaming algorithm for the problem. Additionally, we show that any p-pass algorithm for the problem requires space $\Omega(n/p)$.

2012 ACM Subject Classification Theory of computation → Streaming models; Theory of computation → Fixed parameter tractability; Theory of computation → Streaming, sublinear and near linear time algorithms; Mathematics of computing → Combinatorial algorithms

Keywords and phrases min, ones, sat, d-sat, parameterized, kernelization, streaming, space, efficient, algorithm, parameter

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.8

1 Introduction

The satisfiability problem (SAT) is among most studied NP-complete problems and serves as the canonical problem for NP, being the first problem which was shown to be NP-complete [6, 23]. It is an important problem in both theory and practice, and together with its variants, it appears in nearly every domain of Computer Science (see for example [9, 15, 16, 17, 30]). Because of this, the problem has been studied in various paradigms such as classical Complexity Theory [3], Approximation Algorithms [22, 34], Exact Algorithms [14, 32], Parameterized Complexity [7, 31], and Heuristics [16].

A variant which frequently appears in the literature is d-SAT ($d \geq 1$), where problem instances have at most d variables per clause. While d-SAT is NP-complete for $d \geq 3$, 2-SAT is a classic example of a tractable, i.e. polynomial-time-solvable problem. In this work, we study an optimization version of d-SAT in the framework of parameterized streaming, which combines streaming algorithms and parameterized algorithms.

The streaming framework was formulated to study the behaviour of algorithms that process large amounts of data in a sequential manner. The input appears as a sequence of items and the assumption is that the amount of read-write memory available to the algorithm is very limited, typically logarithmic in the total size of the input. Because of this, the algorithm is unable to store the entirety of its input in memory, and since the input appears in a sequence, the algorithm does not have random access to the it. It may however make multiple passes over the input. The goal in the streaming framework is to process the input by making as few passes (ideally, just one) over it as possible while using as little memory as possible. The study of problems in this framework dates back to the 1980s [12, 28], although the framework was formally established only in 1996 [2, 20]. The other player in the combined framework that we employ is Parameterized Complexity – an approach pioneered by Downey and Fellows [8]. For details on Parameterized Complexity, we refer the reader to the books of Downey and Fellows [8], Flum and Grohe [13], Niedermeier [29], and the recent book of Cygan et al. [7]. Appendix A provides a short introduction to the subject.

Min-Ones d-Sat and the Parameterized Streaming Model. We study the following optimization version of d-SAT which, among other things, generalizes Vertex Cover and d-Hitting Set. For $d \geq 1$, the problem is defined as follows.

Min-Ones d-SAT	**Parameter:** k
Instance: (F, k), where F is a boolean formula with at most d literals per clause, and $k \in \mathbb{N}$.	
Question: Can F be satisfied by setting at most k of its variables to 1?	

It should be noted here that the problem 2-SAT admits a polynomial-time algorithm [4, 10, 25]. Its minimization version however, being a generalization of Vertex Cover is NP-hard [33]. Indeed, the graph in a Vertex Cover instance can be seen as a formula in which the vertices are variables and each edge is a monotone clause containing the two endpoints as (positive) literals.

Fafianie and Kratsch [11] considered the question of kernelizing d-Hitting Set, d-Set Matching and Edge Dominating Set in the streaming model. Chitnis et al. [5] studied the problems Maximal Matching and Vertex Cover in the parameterized streaming model. The space used by these algorithms is $O(f(k)\log n)$, where k is the parameter, n is the size of the input, and $f : \mathbb{N} \to \mathbb{N}$ is a computable function.

The parameterized streaming model relaxes the space constraint of the usual streaming model to $f(k)\log n$, and allows the algorithm to make at most $g(k)$ passes over its input, where $g : \mathbb{N} \to \mathbb{N}$ is a (preferably slowly-growing) computable function. The goal now is to make as few passes over the input as possible, relative to the parameter. Under these new constraints, it is possible to construct streaming algorithms that have more refined space requirements, and we can also perform a more delicate analysis of the streaming complexity of the problem in question. Our results here illustrate this fact.

Our Results. In Section 2, we describe a parameterized streaming algorithm for Min-Ones d-SAT ($d \geq 2$) that solves instances (F, k) using $O\big((kd^{ck} + k^d)\log n\big)$ ($c > 0$, a constant) bits of space and makes $k + 1$ passes. We then show that by carefully simulating the execution stack of the standard branching algorithm for Min-Ones d-SAT, a $(d+1)^k$-pass, $O(k \log n)$-space algorithm can be obtained. We believe that such an approach will be useful in the design of parameterized streaming algorithms for other problems as well. As an application, we show how the two algorithms can be used to solve IP_2 (a restricted Integer Programming problem) in the parameterized streaming model.

Section 3 describes a streaming kernelization for Min-Ones 2-SAT and an application of the algorithm to IP_2. By making $k + 2$ passes over the input formula, it produces a kernel with $O\big(k^6\big)$ clauses while using $O\big(k^6 \log n\big)$ bits of space. It is known that for $d \geq 3$, Min-Ones d-SAT does not admit a polynomial kernel [24] under certain (fairly reasonable) assumptions, ruling out a generalization of this result to larger values of d. Our algorithm also provides an alternative to the known kernelization [27] for the problem, since it can also be executed in the less restrictive random-access machine (RAM) model.

We then exhibit various lower bounds in Section 4 to complement the positive results above. For $d \geq 2$, we show that any k-pass streaming algorithm for Min-Ones-d-SAT requires $\Omega\big(\max\big\{n^{1/k}/2^k, \log(n/k)\big\}\big)$ bits of space in the worst case. This result is obtained by combining a well-known lower bound for the $DISJ_k$ [26] problem from Communication Complexity and a lower bound for the streaming problem POT Pointer Chasing [18]. This (unconditional) lower bound implies, among other things, that the $k + 1$ pass Min-Ones d-SAT ($d \geq 2$) algorithm of section 2 is pass-optimal.

The next result in the section shows that even for $d = 1$, any 1-pass algorithm for Min-Ones d-SAT requires space $\Omega(n)$. This is in contrast to the results of Fafianie and Kratsch [11] and Chitnis et al. [5], who independently showed that there are 1-pass parameterized streaming algorithms for the Vertex Cover problem (a restriction of Min-Ones 2-SAT). Finally, we show that any p-pass algorithm for Min-Ones d-SAT ($d \geq 1$), where p may be a function of both n and k, requires space $O(n/p)$.

▶ Note 1.1. Although we do not provide an explicit accounting of the time used by our algorithms, it is not difficult to see that the streaming FPT algorithms all run in FPT time overall and the kernelizations, in polynomial time.

Related Results. Min-Ones 2-SAT was first studied by Gusfield and Pitt [19], who gave a polynomial-time 2-approximation algorithm for the problem. Misra et al. [27] exhibited an equivalence between Min-Ones-2 AT and Vertex-Cover via a polynomial-time parameter-preserving reduction. Fafianie and Kratsch [11], and Chitnis et al. [5] showed that Vertex Cover admits a single-pass, $\mathrm{O}(k^2)$-space algorithm. As noted earlier, Min-Ones 2-SAT generalizes Vertex Cover. Analogously, Min-Ones d-SAT generalizes d-Hitting Set. The question of kernelizing d-Hitting-Set was studied by Abu-Khzam [1], and Fafianie and Kratsch [11], who gave a single-pass algorithm that produces a kernel with $\mathrm{O}(k^d)$ sets.

Preliminaries. Here we introduce some basic concepts and notation used in the rest of the paper. For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. Let $x \in \{0,1\}^n$ and $i \in [n]$. The i^{th} coordinate of x is denoted by $x[i]$. Consider a set of variables $V = \{x_1, \ldots, x_n\}$. A *literal* is a variable x_i (called an *unnegated* literal) or its negation $\neg x_i$ (called a *negated* literal). A *clause* is a disjunction (OR) of literals, e.g. $(x_1 \vee \neg x_2 \vee \neg x_3)$. It is called *monotone* if it consists entirely of unnegated literals, and is called *anti-monotone* if it consists entirely of negated literals. Clauses containing both negated and unnegated literals are called *non-monotone*.

A conjunction (AND) of clauses is called a CNF *formula*. When each clause has at most d literals, it is called a d-CNF formula. An *assignment* for a CNF formula F over the variable set V is a subset $S \subseteq V$. The assignment satisfies a clause if there is a variable in S that appears unnegated in the clause or a variable in $V \setminus S$ that appears negated in the clause. An assignment which satisfies all clauses in a formula is called a *satisfying* assignment for the formula.

2 Streaming FPT Algorithms

The main result of this section is an algorithm that solves instances (F, k) of Min-Ones d-SAT in $k+1$ passes using space $\mathrm{O}\big((kd^{ck} + k^d)\log n\big)$ ($c > 0$, a constant). We also describe how to simulate the execution of the standard branching algorithm for the problem to solve in instances in $(d+1)^k$ passes using space $\mathrm{O}(k \log n)$ (see Appendix B.1). Using these algorithms as subroutines, we then show how IP_2, a restricted version of the Integer Programming problem, where every constraint has at most two variables, can be solved in the parameterized streaming model (see Appendix D).

The $(k+1)$-pass algorithm begins by a making a single pass over the formula and obtains a set of minimal assignments for certain "essential" monotone clauses in the formula. In the next $k-1$ passes, these assignments are extended as much as possible using the implications appearing in the formula. Finally, the algorithm makes an additional pass to check if the formula as a whole is satisfied by one of the extended assignments.

Let $(\mathcal{F}, k)$ be an instance of Min-Ones 2-SAT on the variable set $V = \{x_1, x_2, \ldots, x_n\}$. The next result shows how a streaming kernelization for d-Hitting Set (defined below) can be used to enumerate minimal solutions for a certain hitting set problem.

d-Hitting Set	**Parameter:** k
Instance: $(U, \mathcal{F}, k)$, where $\mathcal{F}$ is a family of subsets of U of size at most d, and $k \in \mathbb{N}$.	
Question: Is there a set $S \subseteq U$ of size at most k such that $S \cap A \neq \emptyset$ for all $A \in \mathcal{F}$?	

▶ **Proposition 2.1 (♠[1]).** *There is an algorithm Enum-d-HS, that finds the set $\mathcal{S}_k$, of all minimal d-hitting sets of size at most k, for an instance $I = (\mathcal{X}, U, k)$ of d-Hitting Set in time $\mathrm{O}(d^k|I|)$. Moreover, $|\mathcal{S}_k| \in \mathrm{O}(d^k)$ and the algorithm uses space $\mathrm{O}(k|I| + kd^k b_U)$, where $|I|$ is the size of I and b_U is maximum size of the elements of U in bits.*

The following result follows from Observation 1, Theorem 1 and Lemma 7 of [11].

▶ **Proposition 2.2.** *There is a 1-pass streaming algorithm called Stream-HS for d-Hitting-Set, which given an instance $I = (\mathcal{X}, U, k)$ with u_{max} as the maximum element of U, returns an (equivalent instance) $I' = (\mathcal{X}', U' \subseteq U, k)$ using $\mathrm{O}(k^d \log |U|)$ bits of memory and $\mathrm{O}(k^d)$ time at each step, such that the following conditions are satisfied.*

1. *$|\mathcal{X}'| \in \mathrm{O}(k^d)$ and the bit size of I' is bounded by $\mathrm{O}(k^d \log |U|)$.*
2. *Elements of U' are represented using $\log |U|$ bits.*
3. *$S \subseteq U$ (or U') of size at most k is a solution to I if and only if it is a solution to I'.*

We note that in item 1 of Proposition 2.2, the size of I' can be bounded by $\mathrm{O}(k^d \log k)$, by relabeling, but we want to preserve the exact variables, so we do not use relabeling.

Next, we apply the algorithm Stream-HS of Proposition 2.2 to obtain a set, which we call a set of *essential monotone clauses*, $\mathcal{C}_1$, and the set $\mathcal{S}_1$ of all minimal assignments (as sets of variables set to 1) for them of size at most k, as follows.

Pass 1. For each monotone clause $C = (x_1 \vee x_2 \vee \cdots \vee x_{d'})$ (where $d' \leq d$) seen in the stream, pass the set $\{x_1, x_2, \ldots, x_{d'}\}$ to Stream-HS. Let $I_t = (\mathcal{X}_t, U_t, k)$ be the output of Stream-HS once the entire stream has been read. Set $\mathcal{C}_1 = \mathcal{X}_t$. Using Proposition 2.1, compute the set $\mathcal{S}_1$, of all minimal d-hitting sets of size at most k for I_t.

The next lemma bounds the time and the space used in Pass 1.

▶ **Lemma 2.3 (♠).** *Pass 1 uses space $\mathrm{O}((k^d + d^k)k \log n)$ and time $\mathrm{O}(d^k k^d \log n)$ after reading each clause from the stream.*

Let $\mathcal{C}^+$ be the set of all monotone clauses of $\mathcal{F}$, let $\mathcal{F}^+ = \wedge_{C \in \mathcal{C}^+} C$ and $\mathcal{F}_1^+ = \wedge_{C \in \mathcal{C}_1} C$. Recall that $\mathcal{C}_1$ is the set of clauses computed in Pass 1. We have the following observation, which follows from Proposition 2.1 and item 3 of Proposition 2.2.

▶ **Observation 2.4.** $\mathcal{S}_1$ is the set of all minimal satisfying assignments of size at most k for both $\mathcal{F}^+$ and $\mathcal{F}_1^+$.

The next observation relates satisfying assignments to $\mathcal{F}$ and the family $\mathcal{S}_1$.

▶ **Observation 2.5 (♠).** Let $\mathcal{S}$ be the set of all minimal satisfying assignments of size at most k for $\mathcal{F}$. Then for each $S \in \mathcal{S}$, there is $S' \in \mathcal{S}_1$, such that $S' \subseteq S$.

Now we describe the next $k-1$ passes. The algorithm constructs a set $\mathcal{S}_{\mathsf{prm}}$ of prime partial assignments, which will be enough to resolve the instance. Initially, we set $\mathcal{S}_{\mathsf{prm}} = \mathcal{S}_1$.

Pass ℓ ($2 \leq \ell \leq k$). Consider a non-monotone clause $C = (x_1^C \vee x_2^C \cdots \vee x_{d_1}^C \vee \neg y_1^C \vee \neg y_2^C \vee \ldots \neg y_{d_2}^C)$ (where $d_1 + d_2 \leq d$) seen in the stream. For each $S \in \mathcal{S}_{\mathsf{prm}}$, such that $\{y_1^C, y_2^C, \ldots y_{d_2}^C\} \subseteq S$ and $\{x_1^C, x_2^C, \ldots x_{d_1}^C\} \cap S = \emptyset$ we do the following.

- If $|S| = k$, then remove S from $\mathcal{S}_{\mathsf{prm}}$.
- Otherwise, $|S| \leq k-1$. Let $\mathcal{S}'_{\mathsf{prm}} = \mathcal{S}_{\mathsf{prm}}$, and for $i \in [d_1]$, let $S_i = S \cup \{x_i^C\}$. Set $\mathcal{S}_{\mathsf{prm}} = (\mathcal{S}'_{\mathsf{prm}} \setminus \{S\}) \cup \{S_i \mid i \in [d_1]\}$.

[1] Proofs of results marked with a ♠ can be found in the appendices.

Clearly, Pass ℓ, where $2 \leq \ell \leq k$, on reading a clause C uses time $\mathrm{O}(|\mathcal{S}_1|dk)$. Moreover, it modifies the sets in $\mathcal{S}_{\mathsf{prm}}$ (increasing $|\mathcal{S}_{\mathsf{prm}}|$ by at most a factor of d), by either removing a set $S \in \mathcal{S}_1$ completely, or adding one more element to S (when the size is less than k). The above procedure is executed only for $k-1$ passes. Thus, it always maintains that $|\mathcal{S}_{\mathsf{prm}}| \in \mathrm{O}\big(d^{\mathrm{O}(k)}\big)$ (see Proposition 2.1) and each set in $\mathcal{S}_{\mathsf{prm}}$ has at most k elements (each representable by $\log n$ bits). Thus, the (total) space used by the algorithm is bounded by $\mathrm{O}\big((k^d + d^{\mathrm{O}(k)})k \log n\big)$.

For simplicity of description, we introduce the following notation. We set $\mathcal{S}^1_{\mathsf{prm}} = \mathcal{S}_1$ and for each $\ell \in [k]$, we let $\mathcal{S}^\ell_{\mathsf{prm}}$ denote the the set $\mathcal{S}_{\mathsf{prm}}$ after the execution of Pass ℓ. We let $\rho = (Q_1, Q_2, \ldots, Q_t)$ be the sequence of non-monotone clauses in $\mathcal{F}$, where the ordering is given by the order of their appearance in the stream. For $\ell \in [k] \setminus \{1\}$, $i \in [t]$, we let $\mathcal{S}^\ell_{\mathsf{prm}}(i)$ be the set $\mathcal{S}_{\mathsf{prm}}$ (after modification, if any) at Pass ℓ after reading the clause Q_i. Furthermore, we let $\mathcal{S}^\ell_{\mathsf{prm}}(0)$ be the set $\mathcal{S}^{\ell-1}_{\mathsf{prm}}$. Next, we prove some results that will be useful in establishing the correctness of the algorithm.

▶ **Lemma 2.6.** *Let $\mathcal{S}$ be the set of all minimal assignments for $\mathcal{F}$ of size at most k. For each $\ell \in [k]$ and $S \in \mathcal{S}$, there is $S' \in \mathcal{S}^\ell_{\mathsf{prm}}$, such that $S' \subseteq S$.*

Proof. We prove this using induction on ℓ. The claim follows for $\ell = 1$ from Observation 2.5. This forms the base case of our induction. Next, we assume that the claim holds for each $\ell \leq z$ (for some $1 \leq z \leq k-1$) and then we prove it for $\ell = z+1$. At the beginning of ℓth pass when no non-monotone clause is read from the stream, we have for each $S \in \mathcal{S}$, there is $S' \in \mathcal{S}^\ell_{\mathsf{prm}}(0)$, such that $S' \subseteq S$. This follows from the fact that $\mathcal{S}^\ell_{\mathsf{prm}}(0) = \mathcal{S}^{\ell-1}_{\mathsf{prm}}$. Next, we assume that at Pass ℓ, the claim holds after reading the clause Q_i, for each $i \leq p$, where $p \in [t-1] \cup \{0\}$. Now we prove the claim for $Q_{p+1} = (x_1^{p+1} \vee x_2^{p+1} \cdots \vee x_{d_1}^{p+1} \vee \neg y_1^{p+1} \vee \neg y_2^{p+1} \vee \ldots \neg y_{d_2}^{p+1})$. Consider $S \in \mathcal{S}$ and let $\hat{S} \in \mathcal{S}^\ell_{\mathsf{prm}}(p)$, such that $\hat{S} \subseteq S$. We will show that there is a set $S' \in \mathcal{S}^\ell_{\mathsf{prm}}(p+1)$, such that $S' \subseteq S$. Let $X = \{x_1^{p+1}, x_2^{p+1}, \ldots, x_{d_1}^{p+1}\}$ and $Y = \{y_1^{p+1}, y_2^{p+1}, \ldots, y_{d_2}^{p+1}\}$. If $Y \not\subseteq \hat{S}$ or $X \cap \hat{S} \neq \emptyset$, then $\hat{S} \in \mathcal{S}^\ell_{\mathsf{prm}}(p+1)$. Hence, $S' = \hat{S}$ is a set such that $S' \subseteq S$. Otherwise, we have $Y \subseteq \hat{S}$ and $X \cap \hat{S} = \emptyset$. Since S satisfies Q_{p+1}, it must contain a variable, say $x_{i^*}^{p+1}$ from $\{x_1^{p+1}, x_2^{p+1}, \ldots, x_{d_1}^{p+1}\}$. As $X \cap \hat{S} = \emptyset$, $\hat{S} \subseteq S$, $|S| \leq k$, and $x_{i^*}^{p+1} \in S$, we have that $|S| \leq k-1$. For $i \in [d_1]$, let $\hat{S}_i = \hat{S} \cup \{x_i^{p+1}\}$. Recall that $\mathcal{S}^\ell_{\mathsf{prm}}(p+1) = (\mathcal{S}^\ell_{\mathsf{prm}}(p) \setminus \{\hat{S}\}) \cup \{\hat{S}_i \mid i \in [d_1]\}$. From the above we can conclude that $\hat{S}_{i^*} \subseteq S$ and $\hat{S}_{i^*} \in \mathcal{S}^\ell_{\mathsf{prm}}(p+1)$. This concludes the proof. ◀

▶ **Observation 2.7.** For $i \in [k-1]$ and a set $S \in \mathcal{S}^i_{\mathsf{prm}}$, if $S \in \mathcal{S}^{i+1}_{\mathsf{prm}}$, then for each $\ell \in \{i, i+1, \ldots, k\}$, we have $S \in \mathcal{S}^\ell_{\mathsf{prm}}$.

Proof. Consider $i \in [k-1]$ and a set $S \in \mathcal{S}^i_{\mathsf{prm}}$, such that $S \in \mathcal{S}^{i+1}_{\mathsf{prm}}$. Let $\ell \in \{i+2, i+3 \ldots, k\}$ be the lowest integer, such that $S \notin \mathcal{S}^\ell_{\mathsf{prm}}$ (if such an ℓ does not exist, the claim trivially holds). Since $S \in \mathcal{S}^{\ell-1}_{\mathsf{prm}}$ and $S \notin \mathcal{S}^\ell_{\mathsf{prm}}$, there is a non-monotone clause $Q = (x_1 \vee x_2 \cdots \vee x_{d_1} \vee \neg y_1 \vee \neg y_2 \vee \ldots \neg y_{d_2})$, such that $\{y_1, y_2, \ldots, y_{d_2}\} \subseteq S$ and $\{x_1, x_2, \ldots, x_{d_1}\} \cap S = \emptyset$. But we also encountered Q at $(\ell-1)$th pass, and S should have been modified/deleted, which is a contradiction. ◀

▶ **Lemma 2.8.** *Let $\mathcal{S}$ be the set of all assignments for $\mathcal{F}$ of size at most k. For every $S \in \mathcal{S}$, there is $S' \in \mathcal{S}_{\mathsf{prm}}$, such that $S' \subseteq S$ and S' satisfies every clause of $\mathcal{F}$.*

Proof. Consider $S \in \mathcal{S}$ and let $S' \in \mathcal{S}_{\mathsf{prm}} = \mathcal{S}^k_{\mathsf{prm}}$ be a set such that $S' \subseteq S$. The existence of S' is guaranteed by Lemma 2.6. We will show that S' satisfies all the clauses of $\mathcal{F}$. By the construction of $\mathcal{S}_{\mathsf{prm}}$, there is a set $\hat{S} \in \mathcal{S}_1$, such that $\hat{S} \subseteq S'$. Thus, S' satisfies each monotone clause of $\mathcal{F}$ (see Proposition 2.1 and 2.2). Next, consider an anti-monotone clause

$C = (\neg y_1 \vee \neg y_2 \vee \ldots \neg y_{d'})$ (where $d' \leq d$), and let $Y = \{y_1, y_2, \ldots, y_{d'}\}$. Since S satisfies C, $Y_S = Y \setminus S$ is a non-empty set. As $S' \subseteq S$, we have $S' \cap Y_S = \emptyset$. Thus, S' satisfies C. If S' satisfies all the non-monotone clauses of $\mathcal{F}$, then the claim follows. Otherwise, let $C = (x_1 \vee x_2 \cdots \vee x_{d_1} \vee \neg y_1 \vee \neg y_2 \vee \ldots \neg y_{d_2})$ be a non-monotone clause in $\mathcal{F}$ which is not satisfied by S', and let $X = \{x_1, x_2, \ldots, x_{d_1}\}$ and $Y = \{y_1, y_2, \ldots, y_{d_2}\}$. Since S' does not satisfy C, we have $Y \subseteq S'$ and $X \cap S' = \emptyset$. Notice that $Y \subseteq S$ as $S' \subseteq S$. As S satisfies C, we have $S \cap X \neq \emptyset$. This together with the fact that $X \cap S' = \emptyset$ implies that $|S'| \leq k-1$. We can assume that $\hat{S} \neq \emptyset$, as $\mathcal{S}_{\mathsf{prm}}$ can be assumed to contain only non-empty sets, otherwise, $\emptyset$ is a solution to $\mathcal{F}$. The above discussions together with Observation 2.7 and the fact that $|S'| \leq k-1$, implies that $S' \in \mathcal{S}^{k-1}_{\mathsf{prm}}$ (and we have $S' \in \mathcal{S}^{k}_{\mathsf{prm}}$). But then at the kth pass, we would have encountered C, and S' would be replaced by d_1 many sets, namely $S' \cup \{x_i\}$, for each $i \in [d_1]$. This concludes the proof. ◀

In the $(k+1)^{\text{th}}$ pass, the algorithm performs the following steps, whose correctness is established by the discussion above.

Pass $k+1$. Consider a clause C seen in the stream. If there is $S \in \mathcal{S}_{\mathsf{prm}}$, such that S does not satisfy C, then remove S from $\mathcal{S}_{\mathsf{prm}}$. When the stream is over, if $\mathcal{S}_{\mathsf{prm}} \neq \emptyset$, then return yes, and otherwise, return no.

We now have the following theorem.

▶ **Theorem 2.9.** *Instances (F, k) of* Min-Ones d-SAT *($d \geq 2$) can be solved in $k+1$ passes using space $\mathrm{O}\big((kd^{ck} + k^d)\log n\big)$ ($c > 0$, a constant).*

By carefully adapting the standard branching algorithm for Min-Ones-d-SAT, we obtain the following theorem.

▶ **Theorem 2.10 (♠).** *Instances (F, k) of* Min-Ones d-SAT *($d \geq 2$) can be solved in $(d+1)^k$ passes using space $\mathrm{O}(k \log n)$.*

Using Theorem 2.9 and 2.10 we can obtain the following result for IP_2, a restricted Integer Programming problem in which every constraint has at most 2 variables (see Appendix D for details).

▶ **Theorem 2.11 (♠).** IP_2 *admits algorithms that solve instances (P, k) in*
- *$k+1$ passes using space $\mathrm{O}(f(k)\log n)$ ($f : \mathbb{N} \to \mathbb{N}$, a computable function), and in*
- *3^k passes using space $\mathrm{O}(f(k)\log n)$.*

3 Streaming Kernelizations

In this section, we describe a kernelization for Min-Ones 2-SAT that makes $k+2$ passes over instances $(\mathcal{F}, k)$ using space $\mathrm{O}\big(k^6 \log n\big)$ and produces a kernel with $\mathrm{O}\big(k^6\big)$ clauses. In the first pass, the algorithm computes a set of monotone clauses as in Section 2. Then over k more passes, for each variable x appearing in these clauses, the algorithm computes a set of variables which must be set to one if x is set to 1, and the implications that force this. In the last pass, it collects all anti-monotone clauses which only contain variables that also appear in the stored clauses.

We now formally describe our algorithm. Let $(\mathcal{F}, k)$ be an instance of Min-Ones 2-SAT on n variables. In the first pass we apply the algorithm Stream-HS of Proposition 2.2 to obtain a set of monotone clauses, $\mathcal{C}_1$. That is, we do the following.

Pass 1. Obtain a set $\mathcal{C}_1$ of monotone clauses of $\mathcal{F}$ using the same procedure as the first pass of Section 2.

Let V be the set of variables appearing in $\mathcal{F}$, V_1 be the set of variables appearing in $\mathcal{C}_1$. For each variable $v \in V_1$, we maintain a set of variables P_v and a set of clauses $\mathcal{P}_v$. Initially, $P_v = \{v\}$ and $\mathcal{P}_v = \emptyset$, for $v \in V_1$. Now we are ready to describe our next k passes.

Pass ℓ. Consider a non-monotone clause $C = (x \vee \neg y)$ seen in the stream. For each $v \in V_1$ such that $y \in P_v$, $x \notin P_v$, $C \notin \mathcal{P}_v$, and $|P_v| \leq k$, add x and C to the sets P_v and $\mathcal{P}_v$, respectively.

For $v \in V_1$ and $\ell \in [k+1]$, by $P_v(\ell)$ we denote the set P_v at the end of pass ℓ (or at the beginning of pass $\ell+1$, when $\ell = 1$). Furthermore, we let $P = \cup_{v \in V_1} P_v$ and $\mathcal{P} = \cup_{v \in V_1} \mathcal{P}_v$.

▶ **Observation 3.1 (♠).** Let $i \in [k]$ and $v \in V_1$, such that $|P_v(i)| = |P_v(i+1)|$. For all $\ell \in \{i, i+1, \ldots, k+1\}$, we have $|P_v(\ell)| = |P_v(i)|$.

▶ **Lemma 3.2.** *Let S be an assignment which satisfies all clauses in $\mathcal{P}$. For each $v \in V_1 \cap S$, we have $P_v \subseteq S$.*

Proof. Consider $v \in V_1 \cap S$ and let $\rho = (C_1 = (x_1 \vee \neg y_1), C_2 = (x_2 \vee \neg y_2), \ldots, C_t = (x_t \vee \neg y_t))$ be the order in which the clauses in $\mathcal{P}_v$ were added. Note that $P_x = \{x_i \mid i \in [t]\}$. We will show by induction on the index $i \in [t]$ that each $x_i \in S$. Before reading C_1, the only element in P_v was v. As C_1 was added to $\mathcal{P}_v$, it must hold that $y_1 = v$. Since $v \in S$, and S satisfies each clause in $\mathcal{P}$, S must contain x_1. For the induction hypothesis, we suppose that for some $p \in [t-1]$, we have $\{x_i \mid i \in [p]\} \subseteq S$. We will now show that $x_{p+1} \in S$. Since $C_{p+1} \in \mathcal{P}_v$ and C_{p+1} appears after C_i in ρ, for each $i \in [p]$, there exists $z \in \{x_i \mid i \in [p]\}$, such that $z = y_{p+1}$. But since $z \in S$ and S satisfies each clause in $\mathcal{P}$, we have that $x_{p+1} \in S$. ◀

Let $\mathcal{F}'$ be the 2-CNF formula containing all the anti-monotone clauses of $\mathcal{F}$ and all the clauses in $\mathcal{C}_1 \cup \mathcal{P}$.

▶ **Lemma 3.3.** $(\mathcal{F}, k)$ *is a* YES *instance of* Min-Ones 2-SAT *if and only if* $(\mathcal{F}', k)$ *is a* YES *instance of* Min-Ones 2-SAT.

Proof. The forward direction follows from the fact that each clause in $\mathcal{F}'$ is also a clause in $\mathcal{F}$. In the backward direction, let S be a solution to Min-Ones 2-SAT in $(\mathcal{F}', k)$, and $S' = \bigcup_{v \in V_1 \cap S} P_v$. We show that S' is a solution to Min-Ones-2-SAT in $(\mathcal{F}, k)$. Since $V_1 \cap S \subseteq S'$, from Proposition 2.2 we have that S' satisfies each monotone clause of $\mathcal{F}$. From Lemma 3.2 we have $S' \subseteq S$. Thus, S' satisfies each anti-monotone clause of $\mathcal{F}$ ($\mathcal{F}'$ contains all of them). If S' satisfies each non-monotone clause of $\mathcal{F}$, then the claim follows. Otherwise, we have a non-monotone clause $C = (x \vee \neg y)$ in $\mathcal{F}$, which is not satisfied by S'. We have that $x \notin S'$ and $y \in S'$. Let $V_y = \{v \in V_1 \mid y \in P_v\}$. The construction of S' implies that there is $v^* \in V_y$ such that $v^* \in S$. From the construction of S' we have that $x \notin P_{v^*}$. The above discussions together with Observation 3.1 implies that we would have encountered C at a pass $i \leq k$, and we did not add x to P_{v^*}. This means that $|P_{v^*}| \geq k+1$. But this contradicts the fact that S has size at most k (note that from Lemma 3.2 we have $P_{v^*} \subseteq S$). ◀

Let $V_2 = V_1 \cup \left(\bigcup_{v \in V_1} P_v\right)$. We will construct a set $\mathcal{B}$ of anti-monotone clauses. Initially, $\mathcal{B} = \emptyset$. We now describe the $(k+2)^{th}$ pass of our algorithm, which constructs the set $\mathcal{B}$.

Pass $k+2$. For each anti-monotone clause $C = (\neg x \vee \neg y)$ in the stream with $\{x, y\} \subseteq V_2$ and $C \notin \mathcal{B}$, add C to $\mathcal{B}$. Then forget the sets P_v, where $v \in V_1$.

Let $\tilde{\mathcal{F}}$ be the 2-CNF formula obtained from $\mathcal{F}$ by removing all anti-monotone clauses that are not in $\mathcal{B}$.

▶ **Lemma 3.4 (♠).** *$(\mathcal{F}, k)$ is a yes-instance of* Min-Ones-2-SAT *if and only if $(\tilde{\mathcal{F}}, k)$ is a yes-instance of* Min-Ones 2-SAT.

Notice that we have stored the sets of clauses $\mathcal{C}_1$, $\mathcal{P}$, and $\mathcal{B}$, of sizes $\mathrm{O}(k^2)$, $\mathrm{O}(k^3)$, and $\mathrm{O}(k^6)$, respectively. This results in the instance $(\tilde{\mathcal{F}}, k)$ of Min-Ones 2-SAT. The above discussions together with Lemma 3.4 implies the following theorem.

▶ **Theorem 3.5.** Min-Ones-2-SAT *admits an algorithm that kernelizes instances (F, k) in $k+2$ passes using space $\mathrm{O}(k^6 \log n)$ and produces a kernel with $\mathrm{O}(k^6)$ clauses.*

4 Lower Bounds

We begin this section by exhibiting a reduction from the POT Pointer Chasing problem (defined later) to Min-Ones 2-SAT and use it to prove the following theorem.

▶ **Theorem 4.1.** *Any streaming algorithm that solves instances (F, k) of* Min-Ones d-SAT *$(d \geq 2)$ in k passes requires space $\Omega\big(\max\{n^{1/k}/2^k, \log \frac{n}{k}\}\big)$, where n is the number of variables in F.*

The well-known *truncated* disjointness problem of Communication Complexity has the following lower bound.

▶ **Proposition 4.2** (Kushilevitz and Nisan [26], Example 2.12)**.** *Let $n, k \in \mathbb{N}$ with $0 \leq k \leq \lfloor n/2 \rfloor$. Any deterministic protocol for DISJ_k requires $\Omega\big(\log \binom{n}{k}\big)$ bits of communication overall .*

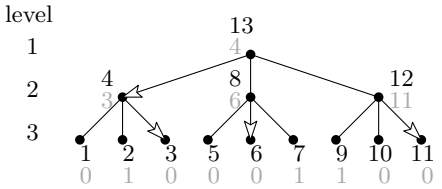
For some background on DISJ_k and other problems (INDEX and DISJ) appearing in the proofs below, the reader is referred to Kushilevitz and Nisan's standard work on Communication Complexity [26].

Using the bound of Proposition 4.2, it is possible to prove the intuitively obvious notion that a streaming algorithm which needs to keep track of locations in its input must use space $\Omega(\log n)$, where n is the size of its input.

▶ **Lemma 4.3.** *Let `MOdSSolve` be a streaming algorithm for* Min-Ones d-SAT *$(d \geq 2)$ that solves instances (F, k) of* Min-Ones d-SAT *on n variables using space $g(n, k)$. For any $k \in \{1, \ldots, \lfloor n/2 \rfloor\}$, if `MOdSSolve` makes p passes to solve instances (F, k), then $g(n, k) = \Omega\big((1/p) \log \binom{n}{k}\big)$.*

Proof. Consider the following protocol for DISJ_k, in which Alice receives the set $S \subseteq \{1, \ldots, n\}$ and Bob receives the set $T \subseteq \{1, \ldots, n\}$ $(|S|, |T| = k)$. Alice constructs the forumla $F_S = \bigwedge_{i \in S} \neg x_i \vee \neg x_i$ and Bob constructs the formula $F_T = \bigwedge_{i \in T} x_i \vee x_i$. Observe that $(F_S \wedge F_T, k)$ is a `YES` instance of Min-Ones 2-SAT if and only if $S \cap T = \emptyset$.

Now alice runs `MOdSSolve` with parameter k and F_S as partial input, and passes its memory r_S to Bob. Bob resumes execution of `MOdSSolve` on the memory r_S and feed it the formula F_T. With this, the algorithm makes the first pass over $F_S \wedge F_T$. Bob then passes the algorithm's memory r_T back to Alice. Using r_T, Alice resumes execution of `MOdSSolve`. The process is repeated for as many passes as the algorithm requires over $F_S \wedge F_T$. Once the algorithm halts, Bob returns its output as his answer.



post-order traversal: 1 2 3 4 5 6 7 8 9 10 11 12 13

stream: 3 3 f(1) f(2) f(3) f(4) f(5) f(6) f(7) f(8) f(9) f(10) f(11) f(12) f(13)

Figure 1 An instance of POT Pointer Chasing with parameters t = 3 and l = 2. The stream consists of t, k and the values of f appearing as in the *lexicographic* post-order traversal of the tree. In the tree, labels appear in black next to vertices, and the corresponding values of f appear in grey. The chain of pointers leads to the vertex labelled 3, with $f(3) = 0$.

Since `MOdSSolve` outputs `YES` if and only if $(F_S \wedge F_T, k)$ is a `YES` instance, the protocol is valid. The amount of communication per pass between Alice and Bob is at most $2g(n,k)$, so the total amount of communication is at most $2pg(n,k)$. From Proposition 4.2, we have $2pg(n,k) = \Omega\big(\log \binom{n}{k}\big)$, i.e. $g(n,k) = \Omega\big((1/p)\log \binom{n}{k}\big)$. ◀

The above result shows an $\Omega(\log n)$ lower bound on the space used by any algorithm that solves instances (F,k) of Min-Ones d-SAT in $\Omega(k)$ passes. This is quite weak, but it is possible to strengthen the result substantially using a lower bound for the following POT Pointer Chasing problem.

Consider a complete t-ary tree T with $l+1$ levels rooted at the vertex r. Let the levels be numbered from 1 to $l+1$, with the root being on level 1. For each non-leaf vertex v, define v_i to be the i^{th} child of v (in the lexicographic ordering of its children). Given a function $f : \mathrm{V}(T) \to \{0,\ldots,t-1\}$, define $f^*(v) = v_{f(v)}$ for non-leaf vertices v and $f^*(v) = f(v)$ for leaf vertices. For $i \in \mathbb{N}$, $(f^*)^i(r)$ denotes the result of applying f^* to r repeatedly, i times.

POT Pointer Chasing

Instance: (T,f), where T is a complete t-ary tree with $l+1$ levels rooted at r, encoded as a post-order traversal of its vertices, and $f : \mathrm{V}(T) \to \{0,\ldots,t-1\}$.
Question: Is $(f^*)^l(r) = 1$?

Figure 1 shows an instance with parameters $t = 3$ and $l = 2$. The following result exhibits a tradeoff between the number of passes made by a streaming algorithm for POT Pointer Chasing and the space it requires.

▶ **Proposition 4.4** (Guha and McGregor [18], Theorem 1). *Any p-pass streaming algorithm that solves* POT Pointer Chasing *instances over t-ary trees with $(p+1)$ levels requires space $\Omega(t/2^p)$ in the worst case.*

▶ **Lemma 4.5.** *Let (T,f) be an instance of* POT Pointer Chasing*, where T is a t-ary tree with $k+1$ levels. A boolean formula F can be constructed such that (T,f) is a `YES` instance of* POT Pointer Chasing *if and only if (F,k) is a `YES` instance of* Min-Ones 2-SAT*.*

Proof. The tree T has levels $1,\ldots,k+1$, with the root r on level 1 and the leaves on level $k+1$. Since each internal vertex has t children, $|V(T)| = \frac{t^{k+1}-1}{t-1} = \mathrm{O}\big(t^k\big)$. Consider the following boolean formula F with $n = \frac{t^k-1}{t-1} = \Theta\big(t^{k-1}\big)$ variables.

Let $w = f^*(r)$, i.e. the $f(r)^{\text{th}}$ child of r, and T_w be the subtree of T rooted at w. The variable set of F is $\{x_v \mid v \in \mathrm{V}(T_w)\}$. For each vertex v on level $i = 2, \ldots, k$ of T, F has the clause $x_v \to x_{f^*(v)} \equiv \neg x_v \vee x_{f^*(v)}$. For each leaf vertex v, F has the clause $\neg x_v \vee \neg x_v$ if and only if $f(v) = 0$. In addition, F has the clause $x_w \vee x_w$.

We now show that (F, k) is an equivalent instance of Min-Ones 2-SAT. Consider the leaf vertex $z = (f^*)^k(r)$, i.e. the vertex reached by chasing pointers from the root of T. If (T, f) is a `YES`-instance, i.e. $f(z) = 1$, then F can be satisfied by setting k variables (corresponding to variables on the w–z path in T) to 1, i.e. (F, k) is a `YES` instance. In the other case, i.e. $f(z) = 0$, F is unsatisfiable: F contains the clause $x_w \vee x_w$, a chain of implications from w to z, and the clause $\neg x_z \vee \neg x_z$, which cannot be satisfied simultaneously. Thus, (F, k) is a `NO` instance. ◀

Observe that the implication $x_v \to x_{f(v)}$ can be produced by simply reading off the value $f(v)$. This is because in the stream, the values of f appear as in the (lexicographic) post-order traversal of T, and knowing the value $f(v)$ and the position of $f(v)$ in the stream is enough to determine the $f(v)^{\text{th}}$ child of v. Thus, the clauses can be produced *on the fly* while making a pass over the post order traversal of T.

We now prove Theorem 4.1.

Proof. Let `MOdSSolve` be a k-pass streaming algorithm for Min-Ones 2-SAT that uses space $g(n, k)$ on inputs (F, k) over n variables. Consider an algorithm that takes as input an instances (T, f) of POT Pointer Chasing over trees with $k+1$ levels, producing instances (F, k) (over $n = \Theta(t^{k-1})$ variables) of Min-Ones 2-SAT on the fly as above, and feeding them as input to `MOdSSolve`. Because of Lemma 4.5, the output of `A` on (F, k) correctly decides (T, f).

The algorithm makes k passes over its input and the amount of space used overall is $\mathrm{O}(g(n, k) + \log n)$. This value is $\Omega(t/2^k)$, by Proposition 4.4. Since $n = \Theta(t^k)$, we have $g(n, k) + \log n = \Omega(n^{1/k}/2^k)$. Consider the case $k \geq \sqrt{\log n}$. The expression $n^{1/k}/2^k$ is $\mathrm{o}(1)$, so $g(n, k) = \Omega(n^{1/k}/2^k)$ holds trivially. In the other case, i.e. $k < \sqrt{\log n}$, we have $g(n, k) = \Omega(\log n)$ by Lemma 4.3, so $g(n, k) + \log n = \mathrm{O}(g(n, k))$, i.e. $g(n, k) = \Omega(n^{1/k}/2^k)$.

Observe that the bound $g(n, k) = \Omega(\log \frac{n}{k})$ holds for any $k \leq \lfloor n/2 \rfloor$ (Lemma 4.3), and for $k > \lfloor n/2 \rfloor$, $g(n, k) = \Omega(\log \frac{n}{k})$ holds trivially. Therefore, we have $g(n, k) = \Omega(\max\{n^{1/k}/2^k, \log \frac{n}{k}\})$. ◀

Suppose a streaming algorithm for Min-Ones 2-SAT uses space $\mathrm{O}(f(k)n^{1/k-\epsilon})$ ($\epsilon > 0$, a constant) to decide instances (F, k) over n variables. Observe that $\lim_{n\to\infty} \frac{f(k)n^{1/k-\epsilon}}{n^{1/k}/2^k} = 0$ for any function f. Thus, we have the following corollary.

▶ **Corollary 4.6.** *Let $\epsilon > 0$ be a number. Any streaming algorithm for* Min-Ones 2-SAT *that uses space $\mathrm{O}(f(k)n^{1/k-\epsilon})$ must make at least $k + 1$ passes over its input.*

The preceding corollary shows that the algorithm of Theorem 2.9, which makes $k + 1$ passes over (F, k), is the best possible inasmuch as the number of passes is concerned. We now exhibit two lower bounds on the space complexity of Min-Ones 2-SAT using Communication Complexity similar to those in Lemma 4.3, which apply to Min-Ones d-SAT even when $d = 1$.

▶ **Theorem 4.7.** *There are no 1-pass streaming algorithms for* Min-Ones d-SAT *($d \geq 1$) that use space $f(k)g(n)$ ($f, g : \mathbb{N} \to \mathbb{N}$, computable functions; $g = \mathrm{o}(n)$) on instances (F, k) with n variables.*

Proof. Observe that any instance (a, b) of INDEX can be encoded as the formula $F = \left(\bigwedge_{a[i]=1} \neg x_i\right) \wedge (x_b)$. $(F, 1)$ is a NO instance if and only if $a[b] = 1$. Suppose there is a 1-pass algorithm for Min-Ones d-SAT that uses space $f(k)g(n)$ on n-variable inputs with parameter k. Alice runs the algorithm on $\bigwedge_{a[i]=1} \neg x_i$ and passes the algorithm's memory to Bob. Bob resumes executing the algorithm on the memory and feeds it the additional clause x_b. Using the output of the algorithm, Bob can determine the value $a[b]$.

It is known that any deterministic 1-pass protocol for INDEX requires $\Omega(n)$ bits of communication (Kushilevitz and Nisan [26], Example 4.19). Because Alice passes the algorithm's memory to Bob, the size of this memory must be $\Omega(n)$, i.e. $f(1)g(n) = \Omega(n)$. Thus, there are no 1-pass parameterized streaming algorithms for Min-Ones d-SAT ($d \geq 1$) that use space $\mathrm{O}(f(k)g(n))$ with $g = \mathrm{o}(n)$. ◀

The above theorem shows that even in the case where every clause consists of exactly one literal, it is not possible to solve an instance of Min-Ones d-SAT in a single pass without using space $\Omega(n)$. Unlike Theorem 4.1, the next result holds in cases where p, the number of passes made by the algorithm, is a more general function of k.

▶ **Theorem 4.8.** *Any p-pass streaming algorithm for* Min-Ones d-SAT *($d \geq 1$) requires space $\Omega(n/p)$.*

Proof. The claim follows from the fact that instances of DISJ can be encoded as SAT formulas in which every clause comprises one literal. Consider the formula $F = \bigwedge(C_S \cup C_T)$, where $C_S = \{x_i \mid i \in S\}$ and $C_T = \{\neg x_i \mid i \in T\}$. $S \cap T = \emptyset$ if and only if F is satisfiable. By standard arguments from Communication Complexity, any p-pass streaming algorithm for Min-Ones 2-SAT must use space $\Omega(n/p)$. ◀

5 Conclusion

In this work, we have proved a variety of results that together provide a complete picture of the parameterized streaming complexity of Min-Ones d-SAT. One of the main results is the streaming algorithm for Min-Ones d-SAT which solves instances (F, k) in $(k + 1)$ passes using space $\mathrm{O}\big((kd^{ck} + k^d) \log n\big)$ ($c > 0$, a constant). The matching $(k+1)$-pass lower bound shows that in terms of the number of passes, this result is the best possible.

It is pertinent to note that such results, i.e. which show a sharp tradeoff between the space complexity of a parameterized streaming problem and the number of passes allowed, are quite scarce in the literature. It would be interesting to see which other parameterized streaming problems exhibit such behaviour.

References

1. Faisal N. Abu-Khzam. Kernelization Algorithms for D-Hitting Set Problems. In *Algorithms and Data Structures*, pages 434–445. Springer Berlin Heidelberg, 2007.
2. Noga Alon, Yossi Matias, and Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. *Journal of Computer and System Sciences*, 58(1):137–147, February 1999.
3. Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
4. Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. *Information Processing Letters*, 8(3):121–123, 1979.

5 Rajesh Chitnis, Graham Cormode, Mohammadtaghi Hajiaghayi, and Morteza Monemizadeh. Parameterized Streaming: Maximal Matching and Vertex Cover. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1234–1251, 2015.

6 Stephen A. Cook. The Complexity of Theorem-proving Procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 1971.

7 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015.

8 Rod G. Downey and Michael R. Fellows. *Fundamentals of Parameterized complexity*. Springer-Verlag, 2013.

9 Dingzhu Du, Jun Gu, Panos M Pardalos, et al. *Satisfiability problem: theory and applications: DIMACS Workshop, March 11-13, 1996*, volume 35. American Mathematical Soc., 1997.

10 Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

11 Stefan Fafianie and Stefan Kratsch. Streaming Kernelization. In *Mathematical Foundations of Computer Science (MFCS)*, pages 275–286, 2014.

12 Philippe Flajolet and G Nigel Martin. Probabilistic counting. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 76–82, 1983.

13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

14 Fedor V. Fomin and Petteri Kaski. Exact Exponential Algorithms. *Commun. ACM*, 56(3):80–88, March 2013.

15 Lance Fortnow. The Status of the P Versus NP Problem. *Commun. ACM*, 52(9):78–86, September 2009.

16 Weiwei Gong and Xu Zhou. A survey of SAT solver. In *AIP Conference Proceedings*, volume 1836, 2017.

17 Jun Gu, Paul W Purdom, John Franco, and Benjamin W Wah. Algorithms for the satisfiability (sat) problem. In *Handbook of Combinatorial Optimization*, pages 379–572. Springer, 1999.

18 Sudipto Guha and Andrew McGregor. Tight Lower Bounds for Multi-Pass Stream Computation Via Pass Elimination. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5125, pages 760–772, 2008.

19 Dan Gusfield and Leonard Pitt. A Bounded Approximation for the Minimum Cost 2-Sat Problem. *Algorithmica*, 8(1-6):103–117, 1992.

20 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. In *External Memory Algorithms, Proceedings of a DIMACS Workshop*, pages 107–118, 1998.

21 Dorit S Hochbaum, Nimrod Megiddo, Joseph (Seffi) Naor, and Arie Tamir. Tight Bounds and 2-Approximation Algorithms for Integer Programs with Two Variables per Inequality. *Mathematical Programming*, 62(1-3):69–83, 1993.

22 David S Johnson. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, 9(3):256–278, 1974.

23 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

24 Stefan Kratsch and Magnus Wahlström. Two Edge Modification Problems without Polynomial Kernels. In *Parameterized and Exact Computation, 4th International Workshop, (IWPEC)*, pages 264–275, 2009.

25 Melven R Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1-2):15–20, 1967.

26 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, New York, NY, USA, 1997.

27 Neeldhara Misra, N. S. Narayanaswamy, Venkatesh Raman, and Bal Sri Shankar. Solving Min Ones 2-Sat as Fast as Vertex Cover. *Theoretical Computer Science*, 506:115–121, 2013.

28 J Ian Munro and Mike S Paterson. Selection and sorting with limited storage. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 253–258, 1978.

29 Rolf Niedermeier. *Invitation to fixed-parameter algorithms*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006.

30 Thomas Stützle, Holger Hoos, and Andrea Roli. A review of the literature on local search algorithms for MAX-SAT. *Rapport technique AIDA-01-02, Intellectics Group, Darmstadt University of Technology, Germany*, 2001.

31 Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 188–202. Springer, 2003.

32 Gerhard J Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization—Eureka, You Shrink!*, pages 185–207. Springer, 2003.

33 Mihalis Yannakakis. Node- and Edge-Deletion NP-Complete Problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 253–264, 1978.

34 Mihalis Yannakakis. On the approximation of maximum satisfiability. *Journal of Algorithms*, 17(3):475–502, 1994.

A A Brief Introduction to Parameterized Complexity

A parameterized problem Π is a subset of $\Gamma^* \times \mathbb{N}$, where Γ is a finite alphabet. An instance of a parameterized problem is a tuple (x, k), where x is a classical problem instance and k is an integer, which is called the *parameter*. The framework of parameterized complexity was originally introduced to deal with NP-hard problems, with the aim to limit the exponential growth in the running time expression to the parameter alone. A central notion in parameterized complexity is *fixed-parameter tractability (FPT)* which means, for a parameterized problem Π, there is an algorithm that given an instance (x, k), decides whether or not (x, k) is a YES instance of Π in time $f(k) \cdot p(|x|)$, where f is a computable function of k and p is a polynomial in the input size. Another central notion in parameterized complexity is *kernelization*, which mathematically captures the efficiency of a data preprocessing. A typical goal of a kernelization algorithm is to store only "small" amount of information, which is enough to recover the answer to the original instance. The "smallness" of the stored information is quantified by the input parameter. Formally, a *kernelization algorithm* or a *kernel* for a parameterized problem Π is given an input (x, k), and the goal is to obtain an equivalent instance (x', k') of Π in polynomial time, such that $|x'| + k' \leq g(k)$. Here, g is some computable function whose value only depends only on k, and depending on whether it is a linear, polynomial, or exponential function, the kernel is called a linear, polynomial, or exponential kernel, respectively. It is well known that a parameterized problem is FPT if and only if it admits a kernel. Thus, in the literature, the term "kernel" is used for polynomial kernels (unless stated otherwise). For more details on parameterized complexity, we refer the reader to the books of Downey and Fellows [8], Flum and Grohe [13], Niedermeier [29], and the recent book by Cygan et al. [7].

B Missing Proofs from Section 2

Proof of Proposition 2.1

The algorithm Enum-d-HS is given in Algorithm 1. We start by proving the correctness of the algorithm by induction on k. When $k \leq 0$, then the algorithm correctly computes the set $\mathcal{S}_k$ (see Steps 1-6). Let us assume that the algorithm returns the correct output for all $k \leq t$, where $t \in \mathbb{N}$. We will now prove that the output of the algorithm is correct for

Algorithm 1 Enum-d-HS.

Input: A set $\mathcal{X}$, of subsets of size at most d of a universe U, and an integer k.
Output: The (multi)set $\mathcal{S}_k$, of all minimal d-hitting sets of size at most k.

```
1  if k < 0 or ∅ ∈ X then
2  |  return ∅;                                   /* no hitting set possible */
3  if k = 0 and there is a non-empty set F ∈ X then
4  |  return ∅;                                   /* no hitting set possible */
5  if k = 0 or there is no set in X then
6  |  return {∅};                                /* ∅ is a hitting set for ∅ */
7  Set S_k = ∅;
8  Let X = {x_1, x_2, ..., x_{d'}} (where d' ≤ d) be an arbitrary non-empty set in X;
9  for i = 1 to d' do
10 |  Let X_i = {Y ∈ X | x_i ∉ Y};
11 |  S^i = Enum-d-HS(X_i, U \ {x_i}, k − 1);
12 |  for each S ∈ S^i do
13 |  |  S_k = S_k ∪ {S ∪ {x_i}};
14 Remove those sets from S_k which are not minimal solutions to (X, U, k);
15 return S_k;
```

$k = t + 1 \geq 1$. If there is no non-empty set in $\mathcal{X}$, then the algorithm returns the correct output (Steps 1-2 and 5-6). Hereafter, we assume that Steps 1-6 are not executed (otherwise, we already have the correct output). Also, we have that $k \geq 1$ and there is a non-empty set $X = \{x_1, x_2, \ldots, x_{d'}\} \in \mathcal{X}$. Any d-hitting set must contain at least one element from X. By induction hypothesis, for each $i \in [d']$, we (correctly) compute the set $\mathcal{S}^i$ of all minimal d-hitting sets of size at most $k-1$, for the instance $(\mathcal{X}_i, U \setminus \{x_i\}, k-1)$. Notice that each set $S \in \mathcal{S}^i$, intersects each set in $\mathcal{X}_i$ and may not intersect X. Moreover, $S \cup \{x_i\}$ is a d-hitting set for $(\mathcal{X}, U, k)$. From the above discussion (together with the induction hypothesis), we obtain that $\mathcal{S}^i_k = \{S \cup \{x_i\} \mid S \in \mathcal{S}^i\}$ is a set containing all minimal d-hitting sets containing x_i for $(\mathcal{X}, U, k)$. Thus, $\cup_{i \in [d']} \mathcal{S}^i_k$ is a set containing all minimal d-hitting sets for $(\mathcal{X}, U, k)$. Moreover, by construction we have that $\mathcal{S}_k = \cup_{i \in [d']} \mathcal{S}^i_k$ with non-minimal solutions removed, is the output returned by the algorithm at Step 17. This concludes the proof of correctness of the algorithm.

We now move to the running time analysis of the algorithm. Notice that the running time of the algorithm is given by the recurrence: $T(k) = d \cdot T(k-1) + \mathrm{O}(|U| + |\mathcal{X}| + |\mathcal{S}_k|)$. Also, the size of $\mathcal{S}_k$ is given by the recurrence $D(k) = d \cdot D(k-1)$, where $0 \leq D(0) \leq 1$. Thus, the running time of the algorithm is bounded by $\mathrm{O}\big(d^k \|I\|\big)$ and $|\mathcal{S}_k| \in \mathrm{O}\big(d^k\big)$. Next, we move to the analysis of the space used by the algorithm. Notice that at any point of time, in the recursive procedure, memory is allocated for at most k copies of Enum-d-HS. Hence, the space required by the algorithm can be bounded by $\mathrm{O}\big(k\|I\| + kd^k b_U\big)$. ◀

Proof of Lemma 2.3

From Proposition 2.2, Pass 1 can compute $I_t = (\mathcal{X}_t, U_t, k)$ after reading all the clauses from the stream using $\mathrm{O}\big(k^d \log n\big)$ space, and using $\mathrm{O}\big(k^d\big)$ time after reading a clause from the stream. Furthermore, $|\mathcal{X}_t| \in \mathrm{O}\big(k^d\big)$, and elements of U_t are represented using $\log n$ bits (by Proposition 2.2 and our assumption that variables of $\mathcal{F}$ are $x_1, x_2, \ldots, x_n$). Now using Enum-d-HS of Proposition 2.1, the algorithm computes $\mathcal{S}_1$ using space (in bits) bounded by $\mathrm{O}\big((k^d + d^k)k \log n\big)$ and time bounded by $\mathrm{O}\big(d^k k^d \log n\big)$. ◀

Proof of Observation 2.5

Any minimal satisfying assignment $S \in \mathcal{S}$ is also a satisfying assignment for $\mathcal{F}^+$. From Observation 2.4 we know that $\mathcal{S}_1$ is the set of all minimal satisfying assignments of size at most k for $\mathcal{F}^+$. Hence, it follows that there is $S' \in \mathcal{S}_1$, such that $S' \subseteq S$. ◀

B.1 $(\mathrm{O}(d^k), \mathrm{O}(k))$-streaming-FPT Algorithm for Min-Ones-d-SAT

In this section, we design an $(\mathrm{O}(d^k), \mathrm{O}(k))$-streaming-FPT algorithm for Min-Ones-d-SAT. The algorithm closely follows the standard $\mathrm{O}(d^k)(n+m)^{\mathrm{O}(1)}$ branching algorithm for Min-Ones-d-SAT, where n and m are the number of variables and clauses in the input instance.

Let $(\mathcal{F}, k)$ be an instance of Min-Ones-d-SAT. By $\mathbb{S}$, we denote the stream of clauses in $\mathcal{F}$. We give our $(\mathrm{O}(d^k), \mathrm{O}(k))$-streaming-FPT algorithm Stream-MOS, for Min-Ones-d-SAT algorithm in Algorithm 2. In the following, we describe various functions of the algorithm Stream-MOS. We note that each of the functions have access to the stream $\mathbb{S}$ and a global variable called pass-count.

1. The function `FinishScan` takes no input and returns no output (only updates pass-count). Its goal is only to read the stream till the end and update pass-count, which stores the number of passes we have made through $\mathbb{S}$. When we enter this function, the pass number is updated. If we are already at the end of the stream $\mathbb{S}$, then it exits without doing any other operation. Otherwise, it read $\mathbb{S}$ till the end and exits. The purpose of defining this function (and maintaining pass-count) is to simplify the analysis of the algorithm.
2. The function `TestSatisfiability` takes as input a set S, and its objective is to determine whether or not S satisfies each clause of $\mathcal{F}$. A call to `TestSatisfiability`, makes a complete scan through $\mathbb{S}$ and we explicitly ensure that whenever it is called, we are at the beginning of the stream. Whenever we find a clause unsatisfied by S in the stream, the function calls `FinishScan` to complete the scanning through remaining clauses of $\mathbb{S}$ and update pass-count, and then it exits after returning 0. In the case when there is no clause which is not satisfied by S, it makes a call to `FinishScan` to update pass-count, and exits after returning 1.
3. The function `FindBranchClause` takes as input a set S. Its objective is to find a clause C which cannot be satisfied (by just) setting variables in S to 1. More precisely, it returns a clause C (if it exists) which satisfies two conditions (to be stated, shortly). Let X and Y be the sets of variables which appear positively and negatively in C, respectively. It must hold that $Y \subseteq S$ and $X \cap S = \emptyset$. Notice that for a satisfying assignment S' for $\mathcal{F}$, such that $S \subseteq S'$, it must hold that $S' \cap X \neq \emptyset$. Moreover, as $S \cap X = \emptyset$, S' must contain at least one more vertex (from X), which is not present in S. We will later see how we use C to progress our branching procedure. To find C, `FindBranchClause` makes a complete scan through $\mathbb{S}$. If it finds a clause C with the desired properties, it makes a call to `FinishScan` to complete the scan through $\mathbb{S}$ and update pass-count, and then it exits after returning C. If a clause with the desired properties is not found even when we reach the end of the stream $\mathbb{S}$, it makes a call to `FinishScan` to update pass-count, and then exits after returning $\diamondsuit$ (indicating that a clause with the desired property could not be found).
4. The function `DetectSolution` takes as input a set S, and its objective is to determine whether or not there is a solution for $(\mathcal{F}, k)$ which sets each variable in S to 1. This function is defined because our algorithm is a recursive procedure, and as the algorithm progresses, we maintain a set of variables that have already been set to 1. We note that at

Algorithm 2 Algorithm Stream-MOS.

Input: A stream of clauses $\mathbb{S}$ for an instance $(\mathcal{F}, k)$ of Min-Ones-d-SAT.

```
pass-count=0;
Function FinishScan()
    pass-count = pass-count+1;
    if at end of the stream 𝕊 then
        return;
    while end of the stream 𝕊 is not reached do
        Read the next clause in the stream;
    return;
Function TestSatisfiability(Set S)
    while end of the stream 𝕊 is not reached do
        Read the next clause C in the stream;
        if C is not satisfied by S then
            FinishScan();
            return 0;
    FinishScan();
    return 1;
Function FindBranchClause(Set S)
    while end of the stream 𝕊 is not reached do
        Read the next clause C in the stream, and let X and Y be the sets of
         variables in C appearing positively and negatively, respectively;
        if Y ⊆ S and S ∩ X = ∅ then
            FinishScan();
            return C;
    return ♢;
Function DetectSolution(Set S)
    if S > k then
        return 0;
    if TestSatisfiability(S)= 1 then
        return 1;
    C = FindBranchClause(S);
    if C ≠ ♢ then
        if |S| = k then
            return 0;
        Let X = {x_1, x_2, ..., x_d'} (where d' ≤ d) be the set of variables appearing
         positively in C;
        ans = 0;
        for i = 1 to d' do
            ans = ans ∨ DetectSolution(S ∪ {x_i});
        return ans;
    return 0;
Function MainMOS()
    res = DetectSolution(∅);
    return res;
```

any point of time we allocate memory only for one such set, and whenever we make calls to other functions, we send the memory location, instead of a separate copy of the set itself. At some steps we call other functions with a modified set (with an element added to S), in this case also we send the memory address after appending the new element (in the front). The above can be achieved by using appropriate memory pointers. Next, we describe the working of `DetectSolution`. If $|S| > k$, then it (correctly) return 0, indicating that there is no satisfying assignment of size at most k containing S. Hereafter, we assume that $|S| \leq k$. Now the function checks if S is a satisfying assignment for $\mathcal{F}$, by making a call to `TestSatisfiability` with (memory location of) S as the argument. If `TestSatisfiability`(S) returns 1, then the function exits after (correctly) returning 1. Otherwise, it makes a call to `FindBranchClause` with (memory location of) S as the argument, and stores the output of it in C. Next, it considers the case when $C \neq \diamondsuit$. Let X and Y be the sets of variables appearing positively and negatively in C, respectively. By the properties of the clauses returned by `FindBranchClause`, we know that $X \cap S = \emptyset$ and $Y \subseteq S$. Thus, for any satisfying assignment S' for $\mathcal{F}$ with $S \subseteq S'$, $S' \cap X \neq \emptyset$ must hold. As $X \cap S = \emptyset$, S' must contain at least one vertex from X and this vertex does not belong to S. If $|S| = k$, then there cannot be a satisfying assignment of size at most k containing S, as otherwise, it will not satisfy C. Thus, in the above case, the function correctly returns 0, and exits. Next, the function deals with the case when $|S| < k$. For any $x \in X$, it checks if there is a satisfying assignment for $\mathcal{F}$ of size at most k containing $S \cup \{x\}$. This is done by making a recursive call to `DetectSolution` with (the memory location of) $S \cup \{x\}$ as the argument. If for any $x \in X$, `DetectSolution`$(S \cup \{x\})$ returns 1, then the function exits after (correctly) returning 1. If for no $x \in X$, `DetectSolution`$(S \cup \{x\})$ returns 1, then the function exits after (correctly) returning 0. If none of the above statements could be used to return an answer, then the algorithm returns 0 and exits.

5. The function `MainMOS` is the main function of the algorithm, where the algorithm begins its execution. The objective of `MainMOS` is to return 1 if $(\mathcal{F}, k)$ is a yes-instance of Min-Ones-d-SAT and return 0, otherwise. Thus, we have only statement, namely, `DetectSolution`$(\emptyset)$ in this function. The correctness of this function follows from the correctness of `DetectSolution`.

Next, we state a lemma regarding Stream-MOS, which will be used to establish the main theorem of this section.

▶ **Lemma B.1.** *Stream-MOS correctly resolves an instance* Min-Ones-d-SAT *(presented as a stream $\mathbb{S}$, of clauses). Moreover, it uses space bounded by $\mathrm{O}(k \log n)$ and makes at most $\mathrm{O}(d^k)$ passes over $\mathbb{S}$.*

Proof. The correctness of Stream-MOS is immediate from the correctness of each of its functions (which is apparent from their respective descriptions). We now bound the space used by the algorithm and the number of passes it makes over $\mathbb{S}$. The space bounds follows from the facts that at any point of the time, we have at most $\mathrm{O}(k)$ active instances of `DetectSolution` and whenever we pass a set as an argument to a function, its memory is passed, rather than a copy of the set itself. To bound the number of passes that the algorithm makes over $\mathbb{S}$, it is enough to bound pass-count. Recall that pass-count is updated only when `TestSatisfiability` or `FindBranchClause` is called by `DetectSolution`. In the above, the pass-count is updated by `TestSatisfiability` or `FindBranchClause` by making a call to `FinishScan`, which increments pass-count exactly by 1. Observe that the total number of (recursive) calls to `TestSatisfiability` or `FindBranchClause`, made by `DetectSolution` is bounded by $\mathrm{O}(d^k)$. Thus, pass-count is bounded by $\mathrm{O}(d^k)$. This concludes the proof. ◀

The proof of Theorem 2.10 follows from Lemma B.1.

C Missing Proofs from Section 3

Proof of Observation 3.1

Consider $i \in [k]$ and $v \in V_1$, such that $|P_v(i)| = |P_v(i+1)|$. Let $\ell \in \{i+2, i+3 \ldots, k+1\}$ be the lowest integer such that $|P_v(\ell)| \neq |P_v(i)|$ (if such an ℓ does not exist, the claim trivially holds). Since $|P_v(\ell-1)| = |P_v(i)|$ and $|P_v(\ell)| \neq |P_v(i)|$, there is a non-monotone clause $Q = (x \vee \neg y)$, such that $y \in P_v(\ell-1)$ and $x \notin P_v(\ell-1)$. But we also encountered C in pass $(\ell-1)$, and P_v should have been modified, which is a contradiction. ◀

Proof of Lemma 3.4

From Lemma 3.3, it is enough to show that $(\mathcal{F}', k)$ is a yes-instance of Min-Ones-2-SAT if and only if $(\tilde{\mathcal{F}}, k)$ is a yes-instance of Min-Ones 2-SAT.

The forward direction follows from the fact that each clause in $\mathcal{F}'$ is also a clause in $\tilde{\mathcal{F}}$. In the backward direction, let S be a solution to Min-Ones 2-SAT in $(\tilde{\mathcal{F}}, k)$. Notice that S satisfies all monotone and non-monotone clauses of $\mathcal{F}'$. For an anti-monotone clause $C = (\neg x \vee \neg y)$, if at least one of x or y is not in V_2, say $x \notin V_2$, then $x \notin S$ (since $S \subseteq V_2$). Otherwise, $x, y \in V_2$, and then C is also a clause in $\tilde{\mathcal{F}}$. Thus, C is satisfied by S. ◀

D Streaming FPT Algorithm for IP_2

In this section, we consider a restriction of the integer programming problem, IP_2 (defined below). We show how to convert an instance of IP_2 to an instance of Min-Ones 2-SAT under parameterized streaming constraints, using the approach of Hochbaum et al. [21]. This allows us to use the algorithms for Min-Ones 2-SAT to solve IP_2. We consider integer programs on n variables and m constraints that have the following form.

$$\text{Minimize } \sum_{j=1}^{n} w_j x_j, \text{ subject to}$$

$$\begin{aligned} a_i x_{p_i} + b_i x_{q_i} &\geq c_i, && (i \in [m], p_i, q_i \in [n]), \\ 0 \leq x_j &\leq u_j, && (j \in [n]), \text{ and} \\ x_j &\in \{0,1\}, && (j \in [n]). \end{aligned}$$

where the coefficients appearing in the constraints are integers, and for all $j \in [n]$, $w_j \in \mathbb{N}$.

Such integer programs (hereafter called *bounded* integer programs) were considered by Hochbaum et al. [21], who showed that by applying a transformation to the variables of the program, the problem of finding a feasible solution becomes equivalent to 2-SAT. We consider the following problem.

IP_2
Input: A bounded-IP $\mathcal{P}$, where we want to minimize $\sum_{j=1}^{n} w_j x_j$, subject to $a_i x_{p_i} + b_i x_{q_i} \geq c_i$, for $i \in [m]$ and $0 \leq x_j \leq u_j$, for $j \in [n]$, and an integer $k \in \mathbb{N}$.
Question: Is there a is feasible solution for $\mathcal{P}$, such that $\sum_{j=1}^{n} w_j x_j \leq k$?

Let $(\mathcal{P}, k)$ be an instance of bounded-IP, where $\mathcal{P}$ is provided as a stream of w_i, for $i \in [n]$, followed by the constraints. As a constraint arrives, we show how we create 2-CNF clauses for it. This will give us an instance of $(\mathcal{F}, k)$, such that $(\mathcal{P}, k)$ is a yes-instance of IP_2 if and only if $(\mathcal{F}, k)$ is a yes-instance of Min-Ones-2-SAT. We note that both the construction and the equivalence of the instances follows from [21], therefore, we only briefly explain the construction of $\mathcal{F}$.

We use the approach described in Section 4 of [21] to construct $\mathcal{F}$. Consider the variable constraint $0 \leq x_p \leq u_p$, for $p \in [n]$. By replacing x_p with u_p binary variables $x_{p,l}$ ($l \in [u_p]$) and introducing the constraints $x_{p,l} \geq x_{p,l+1}$ ($l \in [u_p - 1]$), we obtain an injective correspondence between x_p and $(x_{p,1}, \ldots, x_{p,u_i})$: $x_p = \sum_{l=1}^{u_p} x_{p,l}$. To model these constraints, we add the clause $(x_{p,l} \vee \neg x_{p,l+1})$ to $\mathcal{F}$, for each $l \in [u_p - 1]$.

Let $a_i x_p + b_i x_q \geq c_i$ be a constraint. We only state the case where $a_p, b_q > 0$ (for more details, see [21]). For $i \in [m]$ and $l \in \{0, \ldots, u_p\}$, let $\alpha_{i,l} = \lceil (c_i - l a_i)/b_i \rceil - 1$. The constraint can be expressed by adding the clauses to $\mathcal{F}$ as follows.

- $\left(x_{p,l+1} \vee x_{q,\alpha_{k,l+1}}\right)$, for every $l \in \{0, \ldots u_p - 1\}$ with $0 \leq \alpha_{i,l} < u_q$.
- $x_{p,l+1}$, for every $l \in \{0, \ldots, u_p - 1\}$ with $\alpha_{k,l} \geq u_q$.
- $x_{q,\alpha_{i,l}}$ for $l = u_p$ with $\alpha_{k,u_p} \geq 0$.

Next, we state how weights (and the function to be minimized) are encoded. Note that the weights appearing in the objective function are nonnegative integers. Let x_p be a variable with $w_p > 1$. To express the effect of setting x_p to 1 on the objective function, we introduce $w_p - 1$ additional variables $y_{p,1}, \ldots, y_{p,w_i - 1}$ and the clauses $(\neg x_p \vee y_{p,j})$ to $\mathcal{F}$, for all $j \in [w_i - 1]$.

Producing the clauses as a stream. Under the reasonable assumption that the clauses of $\mathcal{P}$ can each be stored in working memory, i.e. in $\mathrm{O}(f(k) \log n)$ bits of space, and by the construction of $\mathcal{F}$, it is easy to see that as a constraint of $\mathcal{P}$ arrives, we can construct the of corresponding clauses for that constraint in space bounded by $\mathrm{O}(g(k) \log n)$. The above discussions together with the algorithms of Section 2 and B.1, implies the proof of Theorem 2.11.